Hochiminh City University of Technology
Computer Science and Engineering
[CO1027] - Fundamentals of C++ Programming

Operations and Libraries

Lecturer: Duc Dung Nguyen
Credits: 3

Outcomes

- Understand basic components of C++
  - How to use basic operators
  - How to use libraries
  - How to define macro, constants

Today's outline

- Operations
- Libraries functions
- Macro definitions

Basic Operations

- Arithmetic operators: `+, -, *, /, %`
- Bitwise operators: `^, ~, &, |, >>, <<`
- Logic operators: `!, &&, ||, >, <, ==`
- Assignment: `=`

Arithmetic operations

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

Example

```
#include<iostream>
using namespace std;

int main()
{
    cout << 15 / 4 << endl;
    cout << 15 / 4.0 << endl;
    cout << 15 % 4 << endl;
    return 0;
}
```

Assignment operation

Assignment operation

- <left operand> = <expression>
- return <left operand>
- <left operand> can't be constant
- Example:
  - `pi = 3.1415;`
  - `keyPressed = 'q';`
  - `a = b = 5;`

Assignment operation

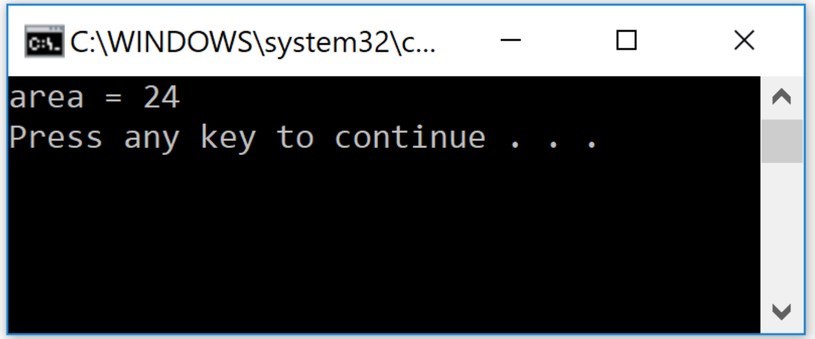
- Assign at the declaration instruction:
  - `int x = 10;`
  - `int y{8};`
  - `float z(10.01f);`

Example

```
#include <iostream>
using namespace std;

int main()
{
   float width = 4.5;
   float height = 5.5;

   int area = width * height;
   cout << "area = " << area << endl;
   return 0;
}
```

Exercise

```
/*Goal: write a program that calculates the volumes of a cube,
** Write the values to the console.
*/
#include<iostream>
using namespace std;

int main()
{
    //Dimension of the cube
    float cubeSide;
    cout << "Enter the size of cube: ";
    cin >> cubeSide;

    //TODO

    return 0;
}
```

Assignment operation

- What is the default type of constants?
- Can we assign different types of value to a variable?

Assignment operation

- Default type of constants depend on how you declare it
  - 10: decimal value, default type depends on context
    - 012: octal value
    - 0x64: hexadecimal value
  - 3.1415: default type is double

Cast operator

- Implicit convert the value from one type to another type
- (<target type>)<expression>
- E.g.:
  - `int a = (int)3.9583;`
  - `float x = (float)a + 0.5f;`
  - `double y = (double)x * (double)a;// y = x * a; is fine`

Implicit conversion

```
int a = 65, integer = 80;
char charA = 65, charB = ‘B’, charC = 67;
float answer = 0, floatNumber = 0.0;

//we can assign an integer to a float
floatNumber = integer;

//we can assign a char to a float
floatNumber = charB;

answer = floatNumber / 4;
//But assigning a float to a char doesn't quite work
charC = answer;

//assigning a float to an interger, results in the float being truncated
integer = answer;
```

Auto type

- *auto type appears from C++11 standard*.
- Should we use auto?
- Where can we use auto?
- auto type: good or bad?

Compound assignments

Operator	Example	Equivalent expression
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Example

PreFix and PostFix

Incrementing

prefix: ++a

postfix: a++

Decrementing

prefix: --a

postfix: a--

Example

```
#include<iostream>
using namespace std;
int main() {
    int a = 10, b = 10;
    int post, pre = 0;
    cout << "Inital values: \t\t\tpost = " << post << " pre= " << pre << "\n";
    post = a++;
    pre = ++b;
    cout << "After one postfix and prefix: \tpost = " << post << " pre= " << pre << "\n";
    post = a++;
    pre = ++b;
    cout << "After two postfix and prefix: \tpost = " << post << " pre= " << pre << "\n";
    return 0;
}
```

C++ Standard Library

C++ Standard Library

- Library is the place where you implement functions, classes to serve some specific tasks.
- Library contains:
  - Definitions: constants, macro, structure, class
  - Functions: implement specific algorithms, a unit of reusable code
  - Class implementations

Library functions

- Function: a named sequence of code that performs a specific task
- Definition
  - `<return type> <function name>(<in/out parameters>);// prototype`
  - ```
    <return type> <function name>(<in/out parameters>)
    {
        // your implementation
    }
    ```
    ```

C++ Standard Library

- Common standard libraries:
  - <stdio.h>, <cstdio>
  - <math.h>, <cmath>
  - <string.h>, <cstring>
  - <assert.h>, <cassert>
  - <errno.h>, <cerrno>
  - <time.h>, <ctime>
- For more detail refer to http://en.cppreference.com/w/cpp/header

<cstdio> library

- `<cstdio>` perform Input/Output operations:
- For more detail refer to http://www.cplusplus.com/reference/cstdio/

Output format

- Text output format
  - `printf("i = %d\n", i);`
  - `cout << "i = " << i << endl;`
- Using function is a convenient way to format output.
- Using I/O streams require a bit modification in the sequence.

Output format

- printf(<format string>, arguments)
  - Format string can contain format specifiers with the following syntax:
    - `%[flags][width][.precision][length]specifier`
    - specifier: d/i, u, o, x/X(uppercase), f/F, e/E, g/G, a/A, c, s, p, n, %(escape character)
    - flags: +, -, space, #, 0
    - .precision: .number, .*
    - width: number, *

Output format

specifier	output	example
d/i	signed decimal integer	-2354
u	unsigned decimal integer	3056
o	unsigned octal	342
x/X	unsigned hexadecimal integer	6f0c
f/F	decimal floating point	3.14159
e/E	scientific notation	3.14159e-05
g/G	use shortest representation	3.14159
a/A	hexadecimal floating point	-0xc.90dep-3
c	character	a
s	string	damn it
p	pointer address	b8000000
n	nothing will be printed, argument must be a pointer to a signed int. The number of printed characters are stored location pointed by the pointer.	
%	print '%' character	%

Output format

```c
#include <stdio.h>

int main()
{
    printf("Characters: %c %c \n", 'a', 65);
    printf("Decimals: %d %ld\n", 1977, 650000L);
    printf("Preceding with blanks: %10d \n", 1977);
    printf("Preceding with zeros: %010d \n", 1977);
    printf("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
    printf("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
    printf("Width trick: %*d \n", 5, 10);
    printf("%s \n", "A string");
    return 0;
}
```

<cstdio> library

```c
/* gets example */
#include <cstdio>

int main()
{
    char string[256];
    printf("Insert your full address: ");
    gets_s(string);
    printf("Your address is: %s\n", string);
    return 0;
}
```


http://www.cplusplus.com/reference/cstdio/gets/

<cmath> library

- `<cmath>` declares a set of functions to compute common mathematical operations :
  - Trigonometric functions (`sin, cos, tan`, etc)
  - Hyperbolic functions (`sinh, cosh, tanh`, etc)
  - Exponential and logarithmic functions (`exp, log`, etc)
  - Power functions (`pow, sqrt`, etc)
  - Rounding and remainder functions (`ceil, floor`, etc)
- For more detail refer to http://www.cplusplus.com/reference/cmath/

<cmath> library

```c
/* sin example */
#include <cstdio>       /* printf */
#include <cmath>        /* sin */

#define PI 3.14159265

int main()
{
    double param, result;
    param = 30.0;
    result = sin(param*PI / 180);
    printf("The sine of %f degrees is %f.\n", param, result);
    return 0;
}
```


http://www.cplusplus.com/reference/cstdio/gets/

Macro definition

Macro definition

- #define/#undef: preprocessor directives
- Extend across single line of code
- No semicolon “;” at the end
- Use “\” to write the define instruction with multiple lines

Macro definition

- Define constants: #define <identifier> <replacement>
  - `#define MAX_LENGTH 50`
  - `#define MY_STRING  “This is a constant string”`
  - `#define pi_2       3.14159/2`
  - `#define pi_2       1.570785`
- Constants variables:
  - `const float x_2 = x / 2;// x_2 cannot be changed`

Macro definition

- More examples:
  - ```
    #define NORMALIZE_FACTOR  50
    …
    float fx = sum / NORMALIZE_FACTOR;
    ```
  - ```
    #define sub(a, b) …
    …
    float x = 0.5 * sub(z + 3.9, y + f(t));
    ```

Macro definition

- Macros:
  - `#define sub(a, b) a - b`
  - `#define sub(a, b) (a - b)`
  - `#define sub(a, b) ((a) - (b))`
  - ```
    #define swap(a, b, c) {\
                    a = b;\
                    b = c;\
                    c = a;\
    }
    ```
    ```

Example

```c
#include <stdio.h>

#define swap(t, x, y) {t tmp = x; x = y; y = tmp;}

int main() {
int x = 10, y = 2;
swap(int, x, y);

printf("%d %d\n", x, y);
return 0;
}
```

Macro definition

- Special operators:
  - #: create a string literal that contains the argument
    - ```
      #define text(a) #a
      …
      cout << text(Be careful) << endl;// print “Be careful”
      ```
  - ##: concatenate two arguments
    - ```
      #define glue(a, b) a ## b
      …
      glue(c, out) << “Weird way to write code\n”;// but acceptable
      ```

Summarise

- Understand basic elements of C/C++
  - Assignment operator, default types, type casting, overflow problem
  - Use library functions
  - Input values
  - Macro, constants